EventADL: Open-Box Anomaly Detection and Localization Framework for Events in Cloud-Based Service Systems

LUAN PHAM, RMIT University, Australia
VICTOR NICOLET, Amazon Web Services, United States
JOEY DODDS, Amazon Web Services, United States
HUI GUAN, Amazon Web Services, United States
DANIEL KROENING, Amazon Web Services, United States

Anomaly detection and localization (ADL) is critical for maintaining reliability and availability in cloud systems. Recent ADL developments focus on metric and log data, leaving event data unexplored. To address this gap, we propose EventADL, the first *open-box* event-based ADL framework for cloud-based service systems. To motivate the design of our framework, we conduct a systematic analysis on 520 real-world incidents, and provide insights into how anomalies and their root causes manifest through event data. EventADL has three phases: offline training, online anomaly detection, and root cause localization. During the training phase, EventADL first learns *Event Semantic Patterns (ESPs)*, which capture normal interactions between system entities using historical event data, and then learns *Event Frequency Patterns (EFPs)*, which capture the normal frequency of known ESPs. In the online anomaly detection phase, any data in the event stream that deviates significantly from either pattern is identified as anomalous. For localization, EventADL constructs an *Intervention Graph* that models the relationships between recent system interactions and the detected anomalies for automatic root cause localization. The framework is designed to operate efficiently with unlabeled data and to produce interpretable anomalies with their corresponding root causes. Our evaluation on three real cloud service systems and two real-world incidents demonstrates that EventADL outperforms existing methods, achieving F1-scores of at least 90% for anomaly detection and 100% top-3 accuracy in root cause localization.

CCS Concepts: • **Software and its engineering → Software creation and management**.

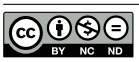
Additional Key Words and Phrases: Anomaly Detection, Root Cause Analysis, Cloud Systems.

ACM Reference Format:
Luan Pham, Victor Nicolet, Joey Dodds, Hui Guan, and Daniel Kroening. 2026. EventADL: Open-Box Anomaly Detection and Localization Framework for Events in Cloud-Based Service Systems. *Proc. ACM Softw. Eng.* 3, FSE, Article FSE179 (July 2026), 23 pages. https://doi.org/10.1145/3808186

1 Introduction

Cloud-based service systems generate large volumes of structured event data that record who performed what operation on which resources and when. These events are captured by monitoring systems to track operations such as API calls, configuration changes, and resource updates. Leading cloud providers offer event monitoring services to support observability and auditing, including AWS CloudTrail [3], Azure Event Hub [34], Google Cloud Audit [16], and Alibaba ActionTrail [2].

Authors' Contact Information: Luan Pham, RMIT University, Australia, luan.pham@rmit.edu.au; Victor Nicolet, Amazon Web Services, United States, victornl@amazon.com; Joey Dodds, Amazon Web Services, United States, jldodds@amazon.com; Hui Guan, Amazon Web Services, United States, huiguan@amazon.com; Daniel Kroening, Amazon Web Services, United States, dkr@amazon.com.



ACM 2994-970X/2026/7-ARTFSE179
https://doi.org/10.1145/3808186

A fundamental challenge faced by cloud operators is **timely anomaly detection and localization (ADL)** from event streams. Anomaly detection identifies unexpected behaviors in the system, while root cause localization (RCL) pinpoints the root causes of those anomalies. Notably, the root cause may not be anomalous on its own. For instance, a resource deletion may seem benign in isolation but could trigger catastrophic failures in downstream services depending on the deleted resource. Our study shows that detecting and diagnosing such incidents often involves multiple teams and significant effort to analyze observable event data. Given the operational scale and high stakes, automated and interpretable ADL for event data is a critical capability in modern cloud [10].

Limitations of Existing Work. While ADL has been actively studied for metrics [9, 17, 18, 28, 41, 42] and unstructured logs [1, 13, 22, 23, 29, 33], event-based ADL remains underexplored. Metric-based methods [28, 41, 42] capture frequency-based anomalies but fail to detect pointwise anomalies (i.e., an individual anomalous event). Log-based methods such as DeepLog [13], LogAnomaly [33], and SwissLog [29] employ deep neural networks (LSTMs and BERT) to capture sequential, quantitative, and temporal patterns in log data. These methods suffer from four key limitations: ❶ they are *closed-box*, offering no interpretability into why an anomaly was flagged; ❷ they perform detection only, without localizing root causes; ❸ they lack adaptive mechanisms for evolving systems, requiring retraining when system behavior changes; and ❹ they ignore structured information readily available in events, such as the interactions between actors and resources (Figure 1).

```
{ "actor.user.name": "merlinary",
  "api.operation": "UpdateInstances",
  "api.request.data": {
    "force": true
  },
  "resources": [
    { "uid": "prod-04242345432" }
  ],
  "cloud.region": "us-east-1",
  "time": "2025-05-19T17:38:32Z",
  "error": null
}
```

Fig. 1. An event in the OCSF schema [38].

Most event-based anomaly detection methods [4, 12, 67] also rely on deep neural networks and thus cannot offer interpretable decisions. ShadeWatcher [67] uses a context-aware embedding model and a GNN to identify anomalous interactions, with anomaly scores derived from deep GNN embeddings. GuardDuty [12] employs a VAE trained on historical data to detect anomalies based on reconstruction error in the latent space. These systems provide no explanation for why an anomaly is detected, requiring further manual investigation. HyGLAD [15] is a pointwise anomaly detection technique which is unable to detect frequency-based anomalies.

Most existing RCL methods focus on metrics, logs, and traces. Nezha [66], MULAN [70], MRCA [60], and CORAL [57] construct causal graphs from metrics, logs, traces and apply centrality or causality analysis for RCL. CONAN [27] extracts contrast patterns from attribute-value pairs to diagnose batch failures. TraceContrast [68] mines contrast sequential patterns from distributed traces. SLIM [47] uses supervised rule-set learning from fault data, and ReACT [48] leverages LLMs to reason over incident reports. While these methods advance RCL in their respective domains, none are designed for structured event data that encodes explicit actor-operation-resource relationships. Their data representations lack the semantic structure that enables open-box detection of *Event Type*, *Event Value*, and *Event Frequency* anomalies, and interpretable RCL through intervention graphs.

Empirical Study. Despite the importance of event data, no prior work has systematically analyzed how anomalies and their root causes manifest through events. We address this gap with a novel empirical study of 520 real-world incident reports from production systems (Section 3). This analysis provides the empirical foundation that has been missing from prior work, offering guidance for designing effective event-based ADL techniques. Our analysis reveals two key findings. First, event-based anomalies manifest along three dimensions: *Event Type (21%)*, *Event Value (68%)*, and *Event Frequency (67%)*. This distribution indicates that effective detection requires capturing both semantic deviations (unusual types or values) and temporal deviations (abnormal frequencies). Second, root causes stem from either a single intervention (*32%*) or multiple interventions (*68%*),

such as resource deletions or configuration changes. This pattern suggests that RCL must trace causal chains from observed anomalies back through potentially multiple triggering events.

Proposed Approach. Guided by these findings, we present EventADL, the first *open-box* ADL framework tailored for event data. EventADL provides both anomaly detection and RCL capabilities while maintaining **interpretability** in two ways. First, when anomalies are detected, operators receive not just alerts but also interpretable explanations that detail *why* these anomalies occurred. Second, all components of EventADL are interpretable and transparent, enabling operators to validate the results and build trust in automated diagnostics. EventADL operates in three phases: offline training, online anomaly detection, and RCL. During offline training, EventADL learns human-interpretable patterns: *Event Semantic Patterns (ESPs)* and *Event Frequency Patterns (EFPs)*. ESPs capture normal event field structures (i.e., Event Types and Values), addressing the Event Type and Event Value anomalies identified in our study. EFPs represent the expected frequency patterns of events, targeting Event Frequency anomalies. During the online detection phase, EventADL detects anomalies by comparing incoming events against learned patterns. Violations of ESPs indicate pointwise anomalies, while deviations from EFPs signal frequency-based anomalies. For RCL, EventADL constructs an *Intervention Graph* capturing interventions that may have triggered the anomalies. The Intervention Graph models causal relationships between actors, operations, impacted resources, and detected anomalies. EventADL applies a time-aware random walk over this graph to identify root causes, tracing the causal chains that our analysis found prevalent.

In summary, we present a comprehensive research effort on event-based ADL, an area that remains underexplored. Our work spans problem formulation, systematic analysis, framework design, extensive evaluation, and dataset release to facilitate future research. Our major contributions are:

- We conduct a systematic analysis of 520 real-world incident reports, offering valuable insights into how anomalies and their root causes manifest in event data in cloud-based service systems.
- We introduce EventADL, the first open-box ADL framework for events in cloud-based service systems. EventADL detects anomalies in Event Type, Event Value, and Event Frequency in event data and localizes their corresponding root causes. Our framework is designed to operate efficiently, with unlabeled data, and to provide an interpretable ADL outcome.
- We extensively evaluate EventADL on three benchmark datasets and two real-world incidents. The results show that EventADL consistently outperforms existing state-of-the-art methods, achieving F1-scores of at least 90% for anomaly detection and 100% top-3 accuracy in RCL.

2 Terminology and Problem Statement

2.1 Terminology

Event. In cloud systems, an event is a structured record that captures at least four key attributes: the `actor` (who performed the action), the `operation` (what action was performed), the `resources` (what was acted upon), and the `timestamp` (when the event occurred). Events may include auxiliary attributes such as execution parameters, metadata, or error indicators. Compared to unstructured logs, which require parsing for analysis [1, 20], events have a known schema. Events are standardized in both open-source [38, 39] and commercial [3, 16, 34] platforms. Figure 1 shows an event following the OCSF schema [38]. *Event Type* refers to the categorical field that identifies the kind of interaction being recorded (e.g., the `api.operation` field in Figure 1, or the `eventName` field in AWS CloudTrail [3]). An Event Type anomaly occurs when an unseen or unexpected interaction type is observed. *Event Value* refers to the attribute values within an event, such as actor identifiers, resource identifiers, or request parameters. An Event Value anomaly occurs when an unusual combination of values appears (e.g., a development user accessing a production database, or an operation

Table 1. How do **anomalies** and their **root cause** manifest through event data in real-world incidents?

ID	Title	Anomaly Symptoms	Root Cause Event
AVA	The Availability of Service X drops in Region A.	The events associated with session token creation in Region A **suddenly disappeared**. There was a significant increase in API call volume (≈14x normal) due to client retries, with **almost all of them (99%) resulting in errors**. No successful token creation was recorded.	The root cause event was **the deactivation of an access key used by Service X** for encryption and signing keys. This deactivation occurred during a credentials cleanup when the credential management tool incorrectly identified used keys.
OUT	Sign-in outage in Region B.	The anomalies were observed as **authentication errors** from AuthService with error messages "Session validation didn't return a principal". The frequency of successful authentication events suddenly stopped for Sign-in service in Region B.	The root cause event was **the deletion of an Identity role** used by Sign-in service to call Service X APIs. This deletion event occurred as part of an infrastructure stack update triggered by the infrastructure pipeline deployment.
AIO	AIOps customers encountered 4XX errors for 3 days while creating AQuery observations.	X events show **a sudden spike of 4XX error responses for API calls**. The errors occurred when users create investigations or observations using AQueryResult events containing newline characters. This resulted in a **96% failure rate in Region A** and **100% failure rate in Region B** for Y-related events.	The root cause event was **the deployment of a code change that introduced an incorrect regex validation pattern** (`^[\S\s]+$` and later `^.*$`) for API input fields, which rejected valid XXLI queries containing newline characters. This change was to address a security requirement.

performed in an unexpected region). In Figure 1, the Event Type is `UpdateInstances`, while Event Values include the actor `merlinary`, resource `prod-04242345432`, and region `us-east-1`.

Anomaly. We view as anomalous any *significant deviation from expected system behavior*. Anomalies manifest in event data through four types: (1) *Event Type*, (2) *Event Value*, (3) *Event Frequency*, and (4) *Event Order*. Although prior studies [7, 55, 67] have examined subsets of these anomalies, none have systematically defined or comprehensively investigated all four. In Section 3, we present an analysis of 520 incident reports to assess the prevalence and significance of these anomaly types.

Root Cause of Anomalies. The root cause of an anomaly refers to the initial event(s) that trigger abnormal behavior. These are typically *intervention events*, such as configuration changes or code deployments. Notably, a root cause may not be an anomaly itself in isolation. For instance, a user deleting a resource may be consistent with past behavior. However, if this deletion disrupts downstream services, it is causally linked to subsequent anomalies.

2.2 Problem Formulation

Let $\mathcal{E} = \langle e_1, e_2, \dots \rangle$ denote a stream of events emitted by a cloud-based service system. Each event e_i is a tuple $e_i = (\Lambda_i, \Omega_i, \mathcal{R}_i, \tau_i)$, where Λ_i is the `actor`, Ω_i is the `operation`, $\mathcal{R}_i$ is the affected `resources`, and τ_i is the `timestamp`. At timestamp t, the system observes a window of recent events denoted by $\mathcal{W}_t = \{e_i \in \mathcal{E} \mid \tau_i \in [t - \Delta, t]\}$, where Δ is the window size. **Anomaly detection** determines whether $\mathcal{W}_t$ contains any anomalous behavior. We define a binary indicator $y \in \{0, 1\}$, where $y = 1$ indicates the presence of anomalies in $\mathcal{W}_t$, and $y = 0$ indicates their absence. If any anomalous event is detected in $\mathcal{W}_t$, then the entire window is considered anomalous. If $y = 1$, the system triggers **RCL** to identify the root cause(s) using a possibly extended window $\mathcal{W}'_t$. Formally, the task is to identify a subset of events $C_t \subseteq \mathcal{W}'_t$ that constitutes the root cause. The final output is the tuple (y, C_t), where y indicates the presence of anomalies and C_t identifies the root cause(s).

3 Analysis of Real-world Incidents

To guide the design of our framework, we conducted a systematic empirical study of incident reports from a large service provider, You-Know-Where (UKW). Each incident report typically contains a high-level summary, detailed information on anomaly symptom(s), mitigation action(s), diagnosed root cause(s), and the incident impact (e.g., the number of affected users, the incident duration). Focusing on how engineers investigate anomalies and use event data during incident diagnosis helps us understand how anomalies and their root causes manifest in production environments.

To systematically select the relevant incident reports, we first queried the incident database using the keywords `"Service X"` and `"X Data"`, where X is UKW's event collection service. The

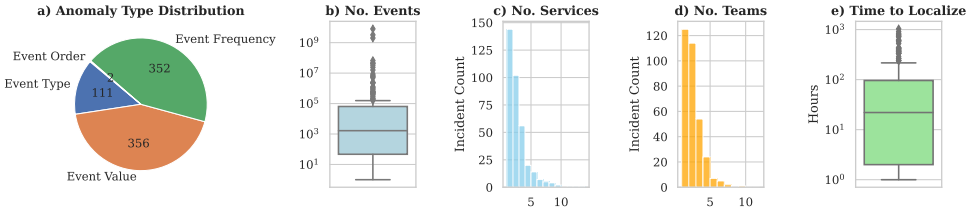


Fig. 2. Insights from real-world incidents. (a) Distribution of anomaly types. (b) Number of events analyzed. (c) Number of services involved. (d) Number of teams involved. (e) Root cause localization time.

search yielded 1,634 incidents mentioning `"Service X"` and 2,928 mentioning `"X Data"` among all incidents since 2013, indicating that event data is a standard part of operational diagnostics. Focusing on recent operational contexts, we restricted our analysis to incidents from the past year (June 2024 – 2025). This search yielded 520 incident reports in total, of which 202 mention `"Service X"` and 354 mention `"X Data"`. Our analysis aims to answer two key questions: **(Q1)** How do anomalies manifest in event data? and **(Q2)** How do root causes leave traces in event data?

3.1 Findings

Figure 2 summarizes our quantitative analysis, and Table 1 provides representative case studies.

Anomaly Symptoms in Event Data (Q1). We categorized anomaly symptoms using the taxonomy introduced in Section 2.1. We observe that: (1) among the four anomaly types, *Event Type*, *Event Value*, and *Event Frequency* dominate the distribution with only 2 incidents exhibiting *Event Order* anomalies and (2) most incidents exhibit more than one anomaly type. Specifically, 111 incidents (21%) involve abnormal Event Type; 356 incidents (68%) involve abnormal Event Value; and 352 incidents (67%) exhibit anomalies in Event Frequency. Event Value anomalies commonly manifest as unusual actor-resource relationships, such as unauthorized actors accessing protected resources, actors operating in unexpected regions, or actors performing operations on resources outside their scope. The two incidents with abnormal Event Order are caused by secret rotation procedures (e.g., deleting the old secret before the new one becomes available). Most incidents involve multiple anomaly types (72%; 375 incidents), and 145 incidents exhibit a single anomaly type.

The most frequent combination is Event Value and Event Frequency (29%). Note that we report the symptoms observed in incident reports; the actual data may sometimes contain more information. For example, we found that the incidents with Event Order anomalies also exhibit anomalies in Event Frequency (e.g., a spike in error events caused by the deleted key). These findings suggest that an anomaly detection solution should monitor for anomalies in Event Type, Event Value and Event Frequency. This would allow the detection of anomalies for all incidents analyzed, and provide a more complete view of anomalies for incidents that combine multiple types.

Root Cause in Event Data (Q2). Root causes are often interventions – that is, explicit operations by actors that alter system resources in ways that trigger downstream issues. Those interventions are often observable in event data. These interventions vary in form: (1) In 32% of incidents, the root cause can be attributed to a single actor performing an operation (e.g., resource deletion). In this case, a single root cause event is sufficient to explain the anomaly. (2) In 68% of cases, root causes involve multiple actors or complex automated workflows (e.g., CI/CD pipelines), and intervention chains consist of many mutating events. Although the root cause can still be captured potentially by a single event, fully understanding it requires tracing through dependent services. (3) Common root causes include resource deletions, misconfigured deployments, and infrastructure updates.

The OUT incident presented in Table 1 is complex, involving a series of events initiated by **a code change**. The code change was merged into the main branch, which triggered an automated CI/CD pipeline, which in turn updated the infrastructure stack and **improperly removed a critical**

Identity role *(root cause)*. As a result, many dependencies lost access to the removed role, leading to **service failures in the sign-in website *(anomaly)***. It took the operators 2 hours to localize the root cause. The incident was mitigated through a manual reactivation of the deleted role. This case highlights the need for interpretable RCL to provide information for detected anomalies, supporting faster incident mitigation.

Q2: How do root causes leave traces in event data?
A: *Root causes frequently manifest as one or more events representing interventions on critical resources. These may be directly attributable to a single event (e.g., a deletion action) or distributed across a causal chain of actions (e.g., deployments).*

3.2 Challenges

Our empirical analysis reveals three major challenges in detecting anomalies and diagnosing their root causes from event data in cloud-based service systems:

C1. Overwhelming Event Volume. The volume of events within a short time window can be overwhelming, making manual analysis difficult. Figure 2b shows that in over 50% of incidents, more than one thousand events needed to be examined, and this number could reach up to one billion. This scale stems from multiple services (Figure 2c gives the number of services involved per incident) and multiple teams (Figure 2d presents the number of teams involved per incident) generating concurrent events, significantly increasing the effort required to localize the root cause. For 71% of incidents, identifying the root cause took more than 10 hours (Figure 2e). This delay prevents timely mitigation. The financial impact can be severe as 81% of incidents cost over $1,000.

C2. Lack of Automatic Event-Based ADL. The detection of incidents largely depends on alerts from impacted downstream services or user reports (93% of incidents). The detection of incidents using event data (7%) typically relies on time-consuming manual efforts from human operators. There is a clear need for proactive, event-based mechanisms that can detect and localize issues before they significantly affect downstream services or user experience.

C3. Lack of Interpretability and Actionability. Even when alarms are triggered from downstream services, they often lack actionable insights. Most systems raise alarms without explaining what went wrong (see Anomaly Symptoms in Table 1). Operators then need to search through large volumes of event data from different sources to find the root causes. There is a strong need for an ADL framework that not only detects anomalies but also provides interpretable and actionable outputs such as root causes, so that operators can understand incidents and respond effectively.

Summary: *Real-world incident analysis highlights the need for an automated, scalable, and interpretable ADL framework. It motivates the design of EventADL, which detects anomalies by modeling event semantics and frequency, and localizes root causes by tracing recent interventions from detected anomalies.*

4 EventADL: Our Proposed Framework

We present EventADL, an open-box ADL framework for event data in cloud-based service systems. EventADL detects three anomaly types (*Event Type*, *Event Value*, and *Event Frequency*), and provides root causes by tracing the causal structure of recent system interventions and detected anomalies.

4.1 Framework Overview

EventADL operates in three phases: (1) Offline Training, (2) Online Anomaly Detection, and (3) RCL, as shown in Figure 3. Each phase operates as follows:

Offline Training. During offline training, EventADL learns two types of interpretable patterns from historical event data. (1) *Event Semantic Patterns (ESPs)* capture semantic relationships across event fields (e.g., `actor`, `operation`, `resource`, and `time`) to represent known behaviors (e.g., normal interactions between system entities). (2) *Event Frequency Patterns (EFPs)* model the frequency

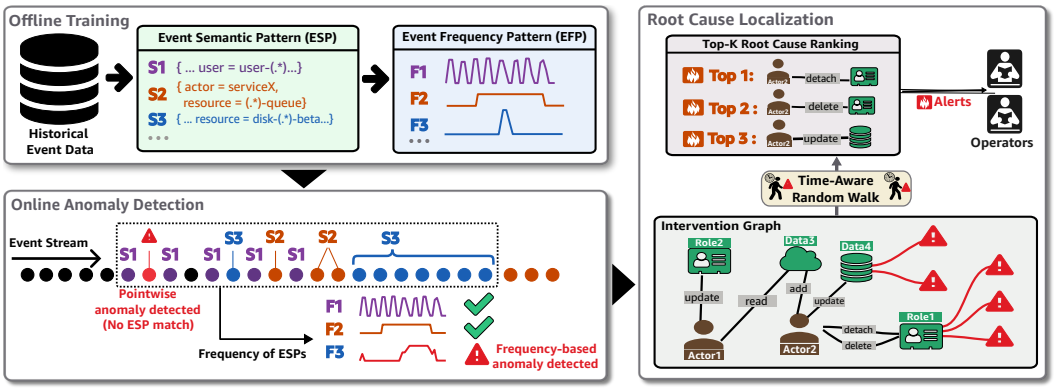


Fig. 3. **Overview of EventADL**. Three phases: offline training (upper left), online anomaly detection (lower left), and RCL (right). Offline, EventADL learns ESPs and EFPs from historical events. Online, events are evaluated against these patterns: ESP identifies pointwise anomalies, while EFP detects frequency-based anomalies. Upon detection, EventADL constructs an Intervention Graph encoding causal links between interventions and anomalies, then applies a time-aware random walk to rank root causes.

of these semantic patterns over time, enabling the detection of frequency-based anomalies. Together, ESPs and EFPs form the foundation for interpretable anomaly detection.

Online Anomaly Detection. As new events arrive, they are continuously compared against the learned ESPs and EFPs. An event is flagged as a *pointwise anomaly* if it does not match any ESP (i.e., an Event Type or Event Value anomaly). A time-window is flagged as a *frequency-based anomaly* if the frequency of ESP-matching events deviates significantly from the corresponding EFP (i.e., an Event Frequency anomaly). These mechanisms capture the three anomaly types observed in practice. Although Event Order anomalies are defined in prior work [55], we found that they are exceedingly rare in practice and often manifest indirectly through frequency anomalies (Section 3).

Root Cause Localization. When an anomaly is detected, EventADL constructs an *Intervention Graph*, which encodes causal relationships between recent system interventions and the detected anomalies. EventADL then performs a time-aware random walk over the graph to identify the root cause(s), i.e., the intervention(s) most likely to be responsible for the detected anomalies. By RCL, EventADL provides interpretable and actionable insights, in contrast to prior approaches [7, 12, 67], which only produce anomaly scores and require further manual investigation.

4.2 Event Semantic Pattern

Event Semantic Pattern (ESP) is a model of the normal event types and values observed in historical event data. They serve two purposes: a set of ESPs is used to detect *pointwise anomalies* (i.e., single events that are anomalous on their own), and they allow labeling normal events with a specific ESP, which is then used to compute EFPs (Section 4.3). *Event Type* and *Event Value* anomalies are pointwise anomalies; we must design ESPs to capture both of those types of anomalies. As for any other component of our system, we also require the ESP model to be interpretable and scalable.

```
{"and": [
 {"==": [{"var": "actor.user.name"}, "merlinary"]},
 {"==": [{"var": "api.operation"}, "UpdateInstances"]},
 {"like": [{"var": "cloud.region"}, "^us-east-[1-4]$"]}
]}
```

Fig. 4. An ESP in the jsonLogic [56] schema.

Formally, the model learned by EventADL on a set of events is a set $\mathcal{S} = \{s_1, s_2, \ldots, s_k\}$ of ESPs, where each ESP s_i is an event-matching rule that captures expected behavior observed in the system. During online detection, if an incoming event $e_j \in \mathcal{E}$ matches any pattern $s_i \in \mathcal{S}$, it is labeled as normal with pattern s_i; otherwise, it is flagged as an anomaly. A rule-based approach provides *interpretability*, *scalability* (Section 5.7), and *deterministic results*.

ESPs are interpretable event matching expressions that capture the events observed in historical data. Figure 4 shows an ESP that captures events where actor `merlinary` performs operation `UpdateInstances` across regions `us-east-[1-4]`. During the online anomaly detection phase, an event with actor `merlinary` performing `UpdateInstances` in `us-west-1` is a pointwise anomaly, as no similar action has been observed before (assuming there are no other patterns matching it). Existing pattern synthesis techniques such as HyGLAD [15] or log parsing techniques such as Drain [20] can extract patterns from event data. In the context of event-based ADL, we show that HyGLAD has a key advantage over Drain: it leverages the structured information in events directly to take into account entity relationships. Table B1 in the supplementary material gives an example where relationship-agnostic methods may mistakenly classify an abnormal event as normal, e.g., {`user:dev-1ac, operation:update, resource:prod-db-1`}. In our experiments, we use HyGLAD to learn ESPs in EventADL, and perform an ablation study using Drain (Section 5.8.3).

4.3 Event Frequency Pattern

Event Frequency Pattern (EFP) detects anomalies in *Event Frequency*, which account for 67% of the analyzed cases (Section 3). From our analysis, we observe that event frequencies are heterogeneous, as they may be dense (e.g., data transfers between services) or sparse (e.g., secret rotation events). We also require EFP to be interpretable and consistent with the open-box design of EventADL.

Our EFP is inspired by the Matrix Profile [32], which provides efficient and interpretable means to characterize frequency patterns through subsequence comparison. However, a key limitation of prior work is its emphasis on the *shape* of subsequences, typically ignoring *magnitude* differences [32]. While shape-based matching may be effective for continuously-valued signals, we argue that *magnitude* is more important in the context of event-based anomaly detection. For instance, a known event pattern occurring at 1–3 requests per second (rps) may appear normal even at 5 rps, but a sudden spike to 100 rps is clearly anomalous. Shape-based anomaly detection methods [32] may fail to detect such deviations (Section 5.8.2). To address this, EventADL uses the Euclidean distance to directly compare subsequences, which is more suitable for event data. EFP is the first adaptation of the Matrix Profile for discrete time series derived from event data.

We construct a set of EFPs for a training period $[t_0, T]$ for each ESP $s_i \in \mathcal{S}$. Let $x_\tau^{(i)}$ denote the frequency of s_i at timestamp τ. Given a subsequence length M, the set of all length-M sliding windows is defined as:

$$W^{(i)} = \left\{ w_u^{(i)} = (x_u^{(i)}, \ldots, x_{u+M-1}^{(i)}) \in \mathbb{N}_0^M \;\middle|\; t_0 \le u \le T - M + 1 \right\}.$$

For each window $w_u^{(i)}$, we compute the minimum Euclidean distance to a non-overlapping window $w_v^{(i)}$, where $|u - v| \ge M$:

$$d_u^{(i)} = \min_{\substack{v=t_0,\ldots,T-M+1 \\ |u-v|\ge M}} \left\| w_u^{(i)} - w_v^{(i)} \right\|_2. \tag{1}$$

We use f_i to denote the EFP with respect to the ESP $s_i \in \mathcal{S}$ as the sequence of nearest distances computed from $W^{(i)}$,

$$f_i = \left\{ d_u^{(i)} \right\}_{u=t_0}^{T-M+1}. \tag{2}$$

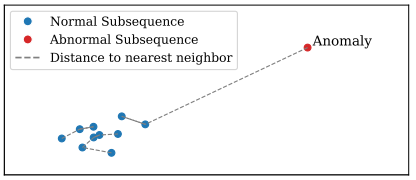


Fig. 5. Detecting anomalies with EFP. Each subsequence $w_u^{(i)}$ is a point in Euclidean space and linked to its nearest non-trivial neighbor $w_v^{(i)}$. The set of distances $\{d_u^{(i)}\}$ forms the Event Frequency Pattern (EFP) $f_i \in \mathcal{F}$. The abnormal subsequence lies far from the cluster of normal subsequences, as it has a *statistically* large distance to its nearest neighbor, indicating a potential anomaly.

The set of all EFPs is denoted by $\mathcal{F} = \{f_1, f_2, \ldots, f_{|\mathcal{S}|}\}$. Each EFP $f_i \in \mathcal{F}$ provides an *interpretable* measure of self-similarity, showing how typical each subsequence is when compared to others (see Figure 5). An anomaly naturally arises when a subsequence has an abnormally large distance to its nearest neighbor, indicating that the observed frequency is *very* different from all known patterns.

To conclude if $w_{\text{new}}^{(i)}$ is anomalous, we perform hypothesis testing on whether $d_{\text{new}}^{(i)}$ is statistically consistent with f_i. Specifically, we treat f_i as samples drawn from an unknown distribution P_D, representing normal frequency of s_i. Then, we define the empirical cumulative distribution function over f_i as $\hat{F}_D(x) = \frac{1}{|f_i|} \sum_{d \in f_i} \mathbf{1}\{d \leq x\}$. The empirical survival function is then given by $\hat{S}(d_{\text{new}}) = 1 - \hat{F}_D(d_{\text{new}})$, which quantifies the probability of observing a distance as large as d_{new}. Let α be a chosen significance level (e.g., $\alpha = 0.01$). If $\hat{S}(d_{\text{new}}) < \alpha$, we reject the null hypothesis that $d_{\text{new}} \sim P_D$ and classify the window as anomalous. This design is distribution-free, making it well-suited to the heterogeneous event frequencies found in large-scale cloud systems.

Robustness to Noise. A key challenge in real-world anomaly detection is the presence of noise (e.g., unknown anomalies) in training data. Many prior methods [6, 7, 13, 15] assume that training data is anomaly-free, which is often unrealistic. In contrast, our approach is robust to such noise because hypothesis testing treats these cases as statistically insignificant. As long as they do not dominate the empirical distribution during normal periods, they have little impact on overall performance. We empirically validate this in Section 5.9.

Adaptation. Cloud systems are constantly evolving with new behaviors, and without adaptation, new behaviors would be repeatedly flagged as anomalies. EventADL's adaptive mechanism works as follows: when an event does not match any ESP, EventADL first flags it as an anomaly then records its count c_s per window. If c_s exceeds a threshold T_s within N successive windows (i.e., persistent), we invoke HyGLAD to revise the ESP set, ensuring only persistent behaviors are added as new normals. For EFPs, EventADL continuously updates f_i with newly observed normal distances $d_u^{(i)}$ (Equation 2), keeping hypothesis testing (Section 4.3) aligned with evolving system characteristics. We note that no online detector can instantly distinguish legitimate system changes from true anomalies without external context (e.g., release notes). Therefore, we also localize root causes to provide interpretable explanations of why the detected anomalies occur (Section 4.4).

4.4 Root Cause Localization

EventADL localizes the root cause to provide interpretable explanations of what triggered the anomaly and why it occurred. Our empirical analysis reveals that many incidents stem from improper interventions that modify one or more system resources, thus leading to anomalies observable in different parts of the system (see Section 3). Motivated by this observation, we introduce the concept of an *Intervention Graph*, which encodes the temporal and causal relationships between system interventions and detected anomalies. This graph can be constructed directly from event data and enables automatic RCL.

Algorithm 1 Construction of the Intervention Graph

Require: Event set $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$
Ensure: Intervention graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}_G)$
Initialize graph $\mathcal{G} \leftarrow (\mathcal{V} \leftarrow \emptyset, \mathcal{E}_G \leftarrow \emptyset)$
for each event $e_i \in \mathcal{E}$ **do**
  Extract `actor`, `operation`, `resources`, `time` from e_i
  Add `actor` node to $\mathcal{V}$ if not already present
  for each `resource` r_j in `resources` **do**
    Add node r_j to $\mathcal{V}$ if not already present
    Add edge (`actor` $\xrightarrow{\texttt{operation,time}}$ r_j) to $\mathcal{E}_G$
    if e_i is associated with an anomaly **then**
      Add `Anomaly` node to $\mathcal{V}$ if not already present
      Add edge ($r_j \xrightarrow{\texttt{time}}$ `Anomaly`) to $\mathcal{E}_G$
    end if
  end for
end for
return $\mathcal{G}$

4.4.1 Intervention Graph. Given a set of structured events $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$ within a time window $\mathcal{W}_t'$, the *Intervention Graph* is a directed multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}_G)$ where $\mathcal{V}$ is the set of nodes representing actors, resources, and anomalies; and $\mathcal{E}_G$ is the set of directed labeled edges representing event-level interactions. For each event e_i involving actor Λ_i, operation Ω_i, time τ_i, and resource set $\{r_1, \ldots, r_k\}$, we add directed edges from Λ_i to each r_j, labeled with (Ω_i, τ_i). If e_i is anomalous, we add an edge from each r_j to an anomaly node, labeled with τ_i (see Algorithm 1).

The resulting graph provides a unified view of how interventions propagate and impact system components (see Figure 3). It enables operators to visually trace anomaly paths and interpret potential root causes. Nevertheless, in practice, large-scale systems generate complex graphs that

are difficult to inspect manually. To this end, we introduce a time-aware random walk algorithm that ranks root cause candidates based on traversal frequency and temporal plausibility.

4.4.2 Time-Aware Random Walk. To automate RCL, we apply a random walk algorithm over the Intervention Graph. The walk begins at each anomaly node and traverses backward through the graph toward potential sources of the anomaly. Nodes visited frequently across multiple walks are identified as root causes. The intuition behind this design is that true root causes often have multiple causal paths leading to observed anomalies, so the random walker will visit them more frequently than unrelated nodes. This assumption is empirically grounded and used in [5, 21, 43, 63]. Our analysis (Section 3) shows that root causes are often interventions that alter system resources, triggering downstream anomalies through multiple causal paths. Our Intervention Graph captures these paths directly from event data, allowing the random walk to visit the root cause frequently from the detected anomalies.

Algorithm 2 Time-Aware Random Walk

Require: Graph G, anomaly nodes A, no. walks N
Ensure: Node-level $visitCount$

Initialize $visitCount \leftarrow \{\}$
for all $(a, t_a) \in A$ **do**
  for $i = 1$ to N **do**
    $u \leftarrow a, t \leftarrow t_a$
    while true do
      $preds \leftarrow$ Predecessors of u in G
      $validEdges \leftarrow [\,]$
      for all $p \in preds$ **do**
        for all edges (p, u, e) in G **do**
          $t_e \leftarrow e$.event_time
          if $t_e \leq t$ **then** ▷ Time-Aware
            Append (p, t_e) to $validEdges$
          end if
        end for
      end for
      if $validEdges = \emptyset$ **then**
        break
      end if
      Sample (p', t') from $validEdges$
      $visitCount[p'] \leftarrow visitCount[p'] + 1$
      $u \leftarrow p', t \leftarrow t'$
    end while
  end for
end for
return $visitCount$

However, a naive traversal may violate causal time constraints. For instance, tracing from an anomaly at 11:00 AM to an operation that occurred at 2:00 PM creates an invalid path. We propose to use a *time-aware random walk* that restricts movement to edges where the associated event occurred no later than the current timestamp within $\mathcal{W}'_t$. This ensures that all paths are temporally valid (see Algorithm 2).

4.4.3 Root Cause Ranking. EventADL returns a ranked list of root causes (i.e., `actor` $\xrightarrow{\texttt{operation,time}}$ `resource`), where the ranking is determined by the visit counts from the time-aware random walk. Interventions that are more frequently visited along valid temporal paths from anomalies are considered more likely to be the root causes. Given this ranked list, operators can focus on the top-ranked interventions instead of manually inspecting all events. Furthermore, the sub-Intervention Graph associated with each ranked root cause can be visualized (Figure 3), reinforcing our open-box design. These visualizations provide interpretable, path-based explanations of how anomalies may have propagated from specific actors or resources through chains of operations.

5 Experiments

We conduct extensive experiments to answer the following questions:

- RQ1: How effective is EventADL in detecting anomalies?
- RQ2: How effective is EventADL in localizing root causes?
- RQ3: How efficient is EventADL?
- RQ4: What is the contribution of each component to EventADL?
- RQ5: How robust is EventADL?

5.1 Benchmark Datasets

At the time of conducting this research, there is no publicly available benchmark for event-based ADL in cloud systems with annotated anomalies and their corresponding root causes. To address this gap, we construct **five benchmark datasets**: three by reproducing incidents on real-world cloud service systems, and two collected from **historical incidents**. We describe both the incident reproductions on benchmark systems and the collected historical incidents in detail below.

5.1.1 Benchmark Systems. We use three service systems (Falcon, Flask and Live) deployed on real infrastructure to reproduce incidents and collect event data. **Falcon** is a cloud-based web application platform consisting of 363 actors and 138,292 resources, designed to host websites. **Flask** is a microservice-based music catalog and retrieval system, comprising 531 actors and 143,353 resources. **Live** is a real-time cricket scoring platform serving 2,507 actors and managing 1,404 resources. Each system generates events across multiple layers (e.g., API calls, configurations, and resource updates). We reproduced three common types of incidents identified in our empirical study. (1) *Secret Deactivation*: We randomly deactivate a secret (e.g., access key), causing permission errors across dependent services and triggering anomalies in event types and frequencies. The root cause is the deactivation event. (2) *Denial-of-Service (DoS)*: We flood API calls, leading to system-wide throttling. Anomalies manifest as spikes in request rates and failure events. The root cause is the actor repeatedly performing the DoS operations. (3) *Unusual Activity*: We reproduce a compromised credential scenario by creating a random resource in an unusual region. These unusual operations are simultaneously anomalies and root causes. We randomly repeat these actions across different resources and permission settings, yielding 30 one-hour test samples per system.

5.1.2 Real-world Incidents. We collected data from two incidents in 2024 at UKW: **OUT** and **AVA** (Table 1). The incident **OUT** was reported after customers were unable to log in to the UKW Management Console or switch between service consoles in Region A for 1h49m. Around 33.1% of the requests failed due to 4xx/5xx errors. The issue stemmed from an improper deletion of a critical role during a code deployment via an infrastructure pipeline. Diagnosing the incident took 100 minutes and involved multiple teams and services. The dataset includes 26,018 actors and 249 resources. EventADL learned 273 ESPs from the normal operation period, reflecting the complex operational patterns in the production environment. The incident **AVA** was reported after 3,355 UKW accounts in Region B experienced ServiceD API failures for 34 minutes, affecting 89 services in total. The root cause was an incorrect deactivation of an access key, due to a software defect and its recent deployments. The issue was resolved by reactivating the key. EventADL learned 308 ESPs for this dataset, capturing the diverse interaction patterns across the affected services.

5.2 Evaluation Metrics

5.2.1 Anomaly Detection. Following existing work [15, 42], we use Precision, Recall, and F1 scores to evaluate the anomaly detectors. When an anomaly detection algorithm successfully detects an abnormal sample (i.e., a case with anomalies), the detection is counted as a True Positive (TP). Conversely, incorrectly classifying an abnormal sample as normal is considered False Negative (FN). Likewise, incorrectly classifying a normal sample as abnormal is considered False Positive (FP). **Precision** is the ratio $\frac{TP}{TP+FP}$, **Recall** is $\frac{TP}{TP+FN}$ and the **F1-score** is determined by $\frac{2\times Precision\times Recall}{Precision+Recall}$.

5.2.2 Root Cause Localization. We use two standard metrics: $AC@k$ and $Avg@k$ to assess the root cause localization (RCL) performance [28, 42]. Herein, we set $k = 1, 3, 5$. Given a set of failure cases A, $AC@k$ is determined by $\frac{1}{|A|}\sum_{a\in A}\frac{\sum_{i<k} R^a[i]\in V^a_{rc}}{min(k,|V^a_{rc}|)}$, and then $Avg@k$ is calculated by $\frac{1}{k}\sum_{j=1}^{k} AC@j$, where $R^a[i]$ denotes the ith ranking result for the failure case a by an RCL method, and V^a_{rc} is the true root cause set of case a. $AC@k$ represents the probability the top k results given by a method include the real root causes. $Avg@k$ measures the overall performance of RCL methods.

5.3 Baselines

We evaluate EventADL against two categories: (i) anomaly detection, and (ii) RCL. For anomaly detection, we include: ADAMAS [17], APE [7], BARO [42], CUSUM [37], DeepSVDD [49], DIF [64], ICL [51], KPIRoot [18], NeuralLog [23], NeuTraL [46], NSigma [28], RCA [30], RDP [58], and ShadeWatcher [67]. For RCL, we compare against: BARO [42], CausalAI [5], CausalRCA [63], DeepHunt [53], ϵ-Diagnosis [50], Groot [59], KPIRoot [18], RCD [21], TVDiag [62], and a *Random*

Table 2. The anomaly detection performance of EventADL and fourteen baselines on five datasets (Falcon, Flask, Live, OUT, AVA) in terms of Precision, Recall, and F1-score. We report the mean and standard deviation over ten runs with different random seeds. We **bold** the best values and underline the second-best.

Method	Falcon			Flask			Live			OUT			AVA		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score
APE [7]	0.46±0.15	0.35±0.16	0.38±0.12	0.47±0.18	0.68±0.28	0.50±0.09	0.36±0.21	0.55±0.28	0.37±0.15	0.16±0.07	**1.00±0.00**	0.27±0.10	0.25±0.09	0.98±0.04	0.39±0.11
BARO [42]	0.18±0.00	**1.00±0.00**	0.30±0.00	0.17±0.00	**1.00±0.00**	0.29±0.00	0.17±0.00	**1.00±0.00**	0.29±0.00	0.06±0.00	**1.00±0.00**	0.12±0.00	0.12±0.00	**1.00±0.00**	0.22±0.00
CUSUM [37]	0.53±0.00	0.84±0.00	0.65±0.00	0.71±0.00	**1.00±0.00**	0.83±0.00	0.77±0.00	**1.00±0.00**	0.87±0.00	0.06±0.00	**1.00±0.00**	0.12±0.00	0.12±0.00	**1.00±0.00**	0.22±0.00
DeepSVDD [49]	0.62±0.45	0.39±0.38	0.44±0.40	0.47±0.46	0.34±0.34	0.34±0.36	0.70±0.44	0.37±0.34	0.43±0.36	0.82±0.10	0.81±0.24	0.79±0.12	0.15±0.04	0.80±0.21	0.25±0.05
DIF [64]	0.15±0.17	0.27±0.41	0.10±0.15	0.21±0.26	0.37±0.38	0.19±0.20	0.18±0.28	0.24±0.37	0.15±0.23	0.43±0.12	0.84±0.24	0.53±0.03	0.10±0.01	0.99±0.01	0.18±0.01
ICL [51]	0.66±0.44	0.39±0.38	0.43±0.40	0.54±0.49	0.36±0.38	0.32±0.35	0.61±0.49	0.22±0.27	0.29±0.32	0.87±0.05	0.99±0.01	**0.92±0.03**	-	-	-
NeuralLog [23]	0.60±0.03	**1.00±0.00**	0.75±0.02	0.42±0.18	0.39±0.21	0.33±0.09	0.53±0.01	**1.00±0.00**	0.69±0.01	0.06±0.00	**1.00±0.00**	0.12±0.00	0.15±0.00	**1.00±0.00**	0.26±0.00
NeuTraL [46]	**0.82±0.34**	0.55±0.39	0.60±0.38	0.68±0.45	0.45±0.38	0.49±0.39	0.68±0.46	0.35±0.36	0.41±0.37	**0.89±0.03**	0.91±0.21	0.89±0.14	-	-	-
NSigma [28]	0.37±0.00	**1.00±0.00**	0.54±0.00	0.17±0.00	**1.00±0.00**	0.29±0.00	0.17±0.00	**1.00±0.00**	0.29±0.00	0.07±0.00	**1.00±0.00**	0.13±0.00	0.16±0.00	**1.00±0.00**	0.28±0.00
RCA [30]	0.40±0.39	0.37±0.41	0.26±0.30	0.30±0.37	0.46±0.41	0.28±0.31	0.27±0.37	0.32±0.40	0.22±0.30	0.55±0.21	0.62±0.22	0.53±0.08	0.14±0.02	0.45±0.12	0.21±0.04
RDP [58]	0.14±0.16	0.80±0.40	0.21±0.23	0.12±0.26	0.80±0.40	0.15±0.28	0.12±0.27	0.50±0.50	0.14±0.28	0.11±0.00	**1.00±0.00**	0.20±0.00	-	-	-
ShadeWat [67]	0.17±0.02	0.76±0.25	0.27±0.03	0.17±0.01	0.82±0.20	0.28±0.03	0.17±0.02	0.90±0.12	0.30±0.03	0.06±0.01	0.90±0.14	0.11±0.01	0.11±0.03	0.83±0.35	0.19±0.06
ADAMAS [17]	0.19±0.00	**1.00±0.00**	0.31±0.00	0.18±0.00	**1.00±0.00**	0.30±0.00	0.17±0.00	**1.00±0.00**	0.29±0.00	0.06±0.00	**1.00±0.00**	0.12±0.00	0.12±0.00	**1.00±0.00**	0.22±0.00
KPIRoot [18]	0.61±0.00	0.71±0.00	0.66±0.00	0.80±0.00	0.53±0.00	0.64±0.00	0.70±0.00	0.53±0.00	0.60±0.00	0.21±0.00	0.40±0.00	0.28±0.00	0.00±0.00	0.00±0.00	0.00±0.00
EventADL	**0.82±0.00**	**1.00±0.00**	**0.90±0.00**	**0.81±0.00**	**1.00±0.00**	**0.90±0.00**	**0.91±0.00**	**1.00±0.00**	**0.95±0.00**	0.81±0.00	**1.00±0.00**	0.90±0.00	**0.91±0.00**	**1.00±0.00**	**0.95±0.00**
ESP-only	0.72±0.00	1.00±0.00	0.84±0.00	0.37±0.00	1.00±0.00	0.54±0.00	0.31±0.00	1.00±0.00	0.47±0.00	0.72±0.00	1.00±0.00	0.84±0.00	0.91±0.00	1.00±0.00	0.95±0.00
EFP-only	0.79±0.00	1.00±0.00	0.88±0.00	0.52±0.00	0.83±0.00	0.64±0.00	0.18±0.00	0.50±0.00	0.26±0.00	0.59±0.00	1.00±0.00	0.74±0.00	1.00±0.00	1.00±0.00	1.00±0.00
EventADL (C)	0.83±0.01	0.96±0.05	0.89±0.03	0.82±0.00	0.93±0.00	0.87±0.00	0.92±0.02	0.90±0.03	0.91±0.01	0.62±0.08	0.93±0.12	0.75±0.09	0.94±0.06	0.93±0.06	0.85±0.18
ESP-only (C)	0.73±0.03	0.78±0.05	0.76±0.04	0.37±0.06	0.81±0.13	0.51±0.08	0.34±0.03	0.88±0.04	0.49±0.03	0.93±0.06	0.70±0.17	0.78±0.10	0.97±0.06	0.87±0.06	0.83±0.16
EFP-only (C)	0.65±0.05	0.83±0.05	0.72±0.05	0.59±0.07	0.78±0.09	0.67±0.08	0.17±0.01	0.40±0.03	0.24±0.01	0.67±0.05	0.87±0.06	0.75±0.01	1.00±0.00	0.83±0.21	0.86±0.20
EventADL (D1)	0.53±0.00	1.00±0.00	0.70±0.00	0.51±0.00	1.00±0.00	0.67±0.00	0.88±0.00	0.93±0.00	0.90±0.00	0.62±0.00	1.00±0.00	0.77±0.00	0.91±0.00	1.00±0.00	0.95±0.00
EventADL (D2)	0.72±0.00	0.94±0.00	0.82±0.00	0.75±0.00	1.00±0.00	0.86±0.00	0.97±0.00	0.93±0.00	0.95±0.00	0.43±0.00	1.00±0.00	0.61±0.00	0.83±0.00	1.00±0.00	0.91±0.00

Note: Methods shown in *italic* are used in the ablation study discussed in Section 5.8. (-) ICL, NeuTraL, and RDP encounter OOM/time-out errors in the AVA dataset.

baseline that selects a root cause at random. We use available implementations when possible. For Groot, APE, and ShadeWatcher, we implement them based on the algorithmic details provided in the original papers. For ADAMAS, DeepHunt, TVDiag, and KPIRoot, we adapt the publicly available code to work with event data. ADAMAS, originally designed for monitoring metrics, is adapted to operate on frequency-based time series derived from ESPs. KPIRoot uses both similarity and causality analysis on the same time series representation. DeepHunt and TVDiag, designed for multimodal data (logs, traces, metrics), are adapted by constructing intervention graphs and feature representations from event data. All hyperparameters follow the recommendations in the respective papers. For methods with configurable thresholds, we follow established practice by running multiple settings and reporting the best performance. Due to space constraints, the detailed description of these baselines are in supplementary material in our replication package.

5.4 Experimental Setting

We conduct all the experiments on Linux servers equipped with 12 CPU and 36 GB RAM. To manage randomness, we repeat each experiment ten times, then report the mean and standard deviation of the collected results. We give each method two hours to finish their tasks, otherwise a *time-out* error is thrown. Our framework is implemented using Python 3.12.

5.5 RQ1: How effective is EventADL in detecting anomalies?

In this RQ, we evaluate the performance of EventADL and the anomaly detection baselines across five datasets. We use the event data during the incident as anomalous samples and event data during normal operation as normal data. We report the average of Precision, Recall, and F1-score over all the cases. Table 2 presents the experimental results, with the best results highlighted in **bold** and second-best results are underlined. We draw the following observations:

(1) EventADL is highly effective across all datasets, outperforming most baselines by large margins. EventADL achieves the highest F1-score on 4 out of 5 datasets and consistently maintains strong performance with F1-score at least 0.90 on all datasets. Because ESPs and EFPs are designed to capture any deviation from training data, EventADL yields 100% recall across benchmarks. The consistent precision above 0.8 shows that ESP and EFP also generalize well over

their training data, avoid false positives. Finally, EventADL shows deterministic performance across runs (i.e., standard deviation = 0), since ESP and EFP training is not affected by randomness.

(2) Metric-based methods achieve high recall scores but suffer from low precision. Metric-based anomaly detectors such as BARO [42], CUSUM [37], NSigma [28], ADAMAS [17], and KPIRoot [18] do not generalize well with event-frequency time series, which exhibit different statistical properties than metrics time series. BARO employs Bayesian Online Change Point Detection for multivariate time series, while CUSUM and NSigma detect deviations in univariate time series based on the mean and standard deviation. These methods detect many anomalies because anomalies in cloud systems often affect multiple entities and manifest as frequency-based deviations (e.g., spikes in error events). BARO models only frequency dependencies with Bayesian statistics, whereas CUSUM and NSigma rely solely on Gaussian assumptions. Such simplifications hinder their ability to model dynamic and heterogeneous behaviors of cloud systems, making them less suitable for complex event-based time series. ADAMAS [17], an adaptive AutoML framework that uses Bayesian Optimization to automatically select and tune its models, achieves 100% recall but low precision. When adapted to event frequencies, its underlying models do not generalize well. KPIRoot [18] achieves moderate F1-scores on benchmark datasets and fails completely on the AVA dataset. KPIRoot uses R-space analysis and ratio-based scoring to detect anomalies in continuous time series like CPU usage and memory utilization. When applied to event-frequency time series with discrete event counts, KPIRoot does not generalize well, leading to inconsistent performance on complex real-world incidents. In contrast, our EFP uses a magnitude-based subsequence distance approach tailored to event-frequency patterns, achieving both high recall and precision.

(3) Deep learning-based methods exhibit moderate performance and poor robustness. Deep learning baselines such as DeepSVDD [49], DIF [64], ICL [51], and RCA [30] achieve lower F1-score than EventADL. They fail to generalize normal event behavior. For instance, RCA trains an ensemble of autoencoders to reconstruct normal data patterns, but autoencoders may degenerate to identity mappings, thereby missing anomalies with low reconstruction loss. More broadly, these methods assume clean training data, which is rarely realistic–unknown anomalies in the training set can significantly degrade performance. For example, DeepSVDD [49] is a one-class classification method that learns to enclose normal data within a hypersphere in latent space, which may also incorrectly generalize anomalies into the learnt hypersphere, leading to low recall as they cannot catch anomalies. In addition, deep learning methods are inherently stochastic, relying on random initialization and sampling, which yields slightly different performance across different runs. This can undermine engineers' trust, as repeated runs lead to inconsistent results. By contrast, EventADL's open-box design and deterministic behavior ensure both accuracy and reliability.

5.6 RQ2: How effective is EventADL in finding the root cause of anomalies?

In this section, we evaluate the performance of EventADL against the existing baselines on all five datasets. We use all events within 1 hour around the anomaly occurrence time for all methods to perform RCL, as we have observed in our benchmark cloud systems and in the collected real-world incidents that the root causes are closely aligned with the anomaly occurrence time. As the cloud systems are highly dynamic, a mistaken operation propagates and affects the systems quickly. We also use the Random baseline as a proxy for a human operator that randomly generates a hypothesis and investigates the root cause, aiming to see how RCL can help increase root cause analysis performance. Table 3 reports the RCL performance of all methods using the AC@1, AC@3, and Avg@5 scores across all five datasets. We run each experiment 10 times with different random seeds and report the mean and standard deviation of their performance. We observe that:

(1) EventADL consistently outperforms all baselines in RCL. It achieves an AC@3 score of 100% on all systems, meaning the true root cause is always ranked among the top three candidates.

Table 3. The anomaly localization performance of EventADL and ten baselines on five datasets (Falcon, Flask, Live, OUT, and AVA) in terms of AC@1, AC@3, and Avg@5. We report the mean and standard deviation over ten runs with different random seeds. We **bold** the best values and underline the second-best.

Method	Falcon			Flask			Live			OUT			AVA		
	AC@1	AC@3	Avg@5	AC@1	AC@3	Avg@5	AC@1	AC@3	Avg@5	AC@1	AC@3	Avg@5	AC@1	AC@3	Avg@5
Random	0.05±0.03	0.22±0.07	0.22±0.04	0.04±0.04	0.20±0.08	0.19±0.06	0.08±0.06	0.24±0.05	0.23±0.05	0.05±0.08	0.20±0.13	0.20±0.11	0.09±0.09	0.24±0.10	0.23±0.09
BARO [42]	0.50±0.00	0.83±0.00	0.73±0.00	0.67±0.00	0.83±0.00	0.81±0.00	0.80±0.00	**1.00±0.00**	0.92±0.00	0.20±0.00	0.20±0.00	0.28±0.00	0.70±0.00	**1.00±0.00**	0.90±0.00
CausalAI [5]	0.18±0.09	0.40±0.10	0.37±0.07	0.14±0.05	0.40±0.06	0.40±0.04	0.24±0.07	0.55±0.07	0.49±0.04	0.00±0.00	0.00±0.00	0.00±0.00	0.07±0.26	0.27±0.45	0.26±0.35
CausalRCA [63]	0.00±0.00	0.67±0.00	0.44±0.00	0.00±0.00	0.00±0.00	0.23±0.00	0.00±0.00	0.60±0.00	0.44±0.00	0.00±0.00	0.00±0.00	0.00±0.00	0.00±0.00	**1.00±0.00**	0.60±0.00
ϵ-Diagnosis [50]	0.03±0.02	0.12±0.06	0.13±0.05	0.03±0.03	0.10±0.05	0.08±0.04	0.00±0.00	0.00±0.00	0.00±0.00	0.04±0.05	0.23±0.12	0.22±0.09	0.00±0.00	0.00±0.00	0.12±0.00
Groot [59]	0.33±0.00	0.33±0.00	0.33±0.00	0.50±0.00	0.50±0.00	0.50±0.00	0.40±0.00	0.40±0.00	0.40±0.00	0.00±0.00	0.00±0.00	0.00±0.00	0.00±0.00	0.00±0.00	0.00±0.00
RCD [21]	0.57±0.00	0.69±0.04	0.65±0.03	0.41±0.03	0.66±0.05	0.60±0.04	0.00±0.00	0.09±0.02	0.11±0.01	0.02±0.04	0.07±0.09	0.05±0.07	0.00±0.00	0.20±0.40	0.14±0.29
DeepHunt [53]	0.53±0.10	0.83±0.00	0.65±0.06	0.53±0.12	0.89±0.07	0.73±0.05	0.82±0.09	**1.00±0.00**	0.90±0.05	0.00±0.00	0.00±0.00	0.00±0.00	0.00±0.00	0.00±0.00	0.04±0.00
TVDiag [62]	0.00±0.00	0.67±0.00	0.40±0.00	0.00±0.00	0.00±0.00	0.00±0.00	0.00±0.00	0.00±0.00	0.00±0.00	-	-	-	-	-	-
KPIRoot [18]	0.43±0.00	0.90±0.00	0.63±0.00	0.13±0.00	0.67±0.00	0.41±0.00	0.60±0.00	0.88±0.00	0.74±0.00	0.00±0.00	**1.00±0.00**	0.50±0.00	0.80±0.00	**1.00±0.00**	0.87±0.00
EventADL	**0.70±0.02**	**1.00±0.00**	**0.91±0.00**	**0.68±0.01**	**1.00±0.00**	**0.91±0.01**	**1.00±0.00**	**1.00±0.00**	**1.00±0.00**	**1.00±0.00**	**1.00±0.00**	**1.00±0.00**	**1.00±0.00**	**1.00±0.00**	**1.00±0.00**

(-) TVDiag [62] requires failure training samples. The OUT and AVA incidents are test data without corresponding failure training samples.

This result validates our assumption that root causes have multiple causal paths to observed anomalies (Section 4.4.2), ensuring the random walker reaches the true root cause. In the Live, OUT, and AVA datasets, EventADL can rank the root causes precisely in the top-1 ranking, achieving an AC@1 of 100%. In the Falcon and Flask datasets, there are a few cases where EventADL could not rank the true root cause (i.e., the actor and their interventions) as the top-1 root cause, because, right before the anomaly occurrence time, multiple actors had interacted with the same set of resources (e.g., multiple users running the same updating stack), causing confusion for the random walk backtracking. However, EventADL narrows down the search space by providing a small ranked list of root causes for investigation. It is worth noting that EventADL is the first RCL method designed to work directly with event data as defined in Section 2.1. Therefore, it can capture the operations performed in the systems and precisely construct the intervention graph (Algorithm 1). This graph subsequently allows our time-aware random walk to localize the root cause automatically and precisely. Meanwhile, all existing RCL methods for cloud systems [1, 10, 52] are developed for metrics, logs, and traces, which undermines the power of event data and leaves it under-explored.

(2) Metric-based RCL methods show reasonable performance but fail to fully exploit event data. BARO and ϵ-Diagnosis are metric-based RCL methods. To apply them, we transform event data into time series reflecting the frequency of actor interventions, where the number of series depends on the number of actors active during the incident. BARO consistently achieves the second-best performance. Its core assumption is that the root cause exhibits a strong anomaly during the incident (i.e., the most anomalous time series is considered the root cause). This assumption is surprisingly effective, but does not generalize as BARO performs poorly on the OUT dataset, where multiple rare interventions occurred concurrently, confusing the algorithm as BARO does not correlate interventions with anomalies as in EventADL. Moreover, BARO provides little interpretability, offering no explanation for why a candidate is flagged as the root cause, thereby shifting the burden of reasoning to operators. In contrast, EventADL does not rely on such assumptions. Instead, it constructs an intervention graph that explicitly encodes the causal relationships between recent interventions and detected anomalies, enabling precise and interpretable RCL.

(3) Causal inference methods fail to capture semantic relationships in event data. CausalAI, CausalRCA, RCD, and KPIRoot are causal inference–based RCL techniques, originally designed for metric time series. CausalAI, CausalRCA, and RCD apply causal discovery algorithms (e.g., RCD with Ψ-PC, CausalRCA with DAG-GNN) on the time series data to construct causal graphs under the assumption that the root-cause time series influences many others. Root cause ranking is then performed via graph centrality algorithms such as PageRank. KPIRoot [18] extends this paradigm by integrating both similarity analysis using Jaccard coefficient and causality analysis using Granger causality to identify sequential dependencies between KPI metrics. When adapted

to event data, we transform events into frequency-based time series. KPIRoot shows inconsistent results as it achieves AC@3 scores between 0.67 and 0.90 on benchmark datasets but zero AC@1 on the OUT dataset. Similarly, Groot [59] constructs an event causality graph and applies PageRank to identify high-impact nodes. However, Groot achieves limited accuracy with AC@3 ranging from 0.33 to 0.50 on benchmark datasets and 0.00 on real-world incidents, as it relies on predefined monitoring events rather than directly modeling actor interventions.

Despite the sophistication of these methods, they all share a fundamental limitation as they operate exclusively on time series representations, which fail to capture the semantic relationships between actors, operations, and resources encoded in events. For example, Granger causality identifies temporal precedence between time series but cannot reason about intervention logic: *which* actor performed *what* operation on *which* resource. By contrast, the Intervention Graph in our EventADL leverages event information directly, explicitly encodes the relationships presented in event data that enable precise RCL through time-aware random walk.

(4) Existing graph-based RCA approaches for microservices fail due to the mismatch between microservice dependency graphs and event-based intervention graphs. DeepHunt [53] and TVDiag [62] represent recent advances in microservice failure diagnosis. DeepHunt uses graph autoencoders (GAE) trained in a self-supervised manner on normal system behavior to learn patterns from metrics, logs, and traces. It computes root cause scores by integrating reconstruction errors with failure propagation patterns in service dependency graphs. TVDiag employs task-oriented learning with contrastive cross-modal associations to extract view-invariant failure information, but requires labeled historical failure samples from the target system for supervised training. This training data requirement becomes a critical limitation as the OUT and AVA real-world incidents serve as test data without corresponding failure training samples, preventing TVDiag from being evaluated. DeepHunt can be applied to these incidents by training on normal data from the same systems. Furthermore, TVDiag's architecture assumes a fixed system topology, making transfer learning from benchmark systems to real-world incidents infeasible.

Both methods face a fundamental mismatch when applied to event-based ADL. They are designed to operate on *service dependency graphs*, constructed from traces. In contrast, the *intervention graphs* constructed from events model actor–operation–resource relationships, fundamentally different semantics from service dependencies. This semantic mismatch causes severe generalization failures. DeepHunt achieves an AC@3 score of 0.83 on the Falcon dataset but drops to 0.00 AC@3 on both OUT and AVA incidents. Although self-supervised learning allows DeepHunt to avoid labeled failure data, it still requires normal training data from each target system. TVDiag compounds this limitation by requiring labeled failure samples from the target system, therefore, it cannot be evaluated on OUT and AVA incidents, which lack such training data. In contrast, EventADL's unsupervised, intervention-based approach requires no training data and works across all system layers by directly modeling the event semantics, achieving 100% AC@1 on both real-world incidents.

5.6.1 Failure Case Analysis. We analyze failure cases where the root cause was not ranked first. The dominant failure pattern occurs when background services (e.g., logging or monitoring agents) interact with many resources within the observation window. These services create numerous edges in the Intervention Graph, forming high-connectivity nodes that accumulate more random walk visits than the root cause. In these cases, the ground truth actor typically interacts with a few resources, resulting in fewer paths for the random walker. Despite this limitation, EventADL effectively narrows the search space from potentially hundreds of actors to just three candidates. Combined with the interpretable Intervention Graph, operators can quickly recognize the root cause by examining its sequence of operations and their relationships to the detected anomalies.

5.7 RQ3: How Efficient is EventADL?

Table 4 reports the runtime performance of EventADL against the baselines. All methods are evaluated over 10 runs across all datasets and we report their mean and standard deviation. For fairness, we report only the inference time for deep learning methods (e.g., NeuralLog, APE), excluding their significant training overheads.

5.7.1 Anomaly Detection. As shown in Table 4 (a), **EventADL is highly efficient**, consistently outperforming most baselines across all datasets. This efficiency can be attributed to the design of its two modules: ESP, which relies on lightweight event-matching expressions, and EFP, which uses subsequence distance comparison. NSigma is extremely efficient due to its simplicity as it only considers the deviation from mean and standard deviation of the frequency. In contrast, deep learning methods are significantly slower, even though we report only their inference time. These methods require substantial computational resources during both training and inference.

Table 4. Runtime comparison of EventADL and baselines.

(a) Anomaly Detection Runtime (in seconds).

Method	Falcon	Flask	Live	OUT	AVA
APE [7]	0.060±0.08	0.075±0.08	0.141±0.14	0.093±0.02	1.290±0.19
BARO [42]	0.886±0.08	0.913±0.06	0.890±0.06	0.883±0.06	0.830±0.06
CUSUM [37]	0.011±0.04	0.010±0.04	0.003±0.00	0.010±0.04	0.016±0.05
DeepSVDD [49]	7.780±3.59	10.25±7.01	9.500±7.74	5.560±2.12	5.416±1.75
DIF [64]	46.22±46.5	95.03±130	86.54±117	12.68±1.39	17.31±1.38
ICL [51]	110.2±144	245.9±404	229.3±329	27.34±3.56	-
NeuralLog [23]	4.642±2.83	1.375±0.11	0.971±0.27	0.338±0.02	3.820±0.21
NeuTraL [46]	11.32±6.88	16.79±15.6	18.09±20.3	9.290±0.56	-
NSigma [28]	0.003±0.00	0.003±0.00	0.003±0.00	0.012±0.01	0.006±0.01
RCA [30]	18.91±15.1	32.75±39.6	31.25±39.2	12.38±5.30	143.0±1.91
RDP [58]	8.440±3.94	11.37±8.33	10.64±8.99	9.860±5.17	-
ShadeWat [67]	0.147±0.75	0.131±0.71	0.047±0.22	0.010±0.01	0.570±0.18
ADAMAS [17]	4.847±0.98	5.547±0.99	4.599±0.70	2.839±0.26	92.89±1.52
KPIRoot [18]	0.040±0.01	0.022±0.01	0.041±0.02	0.030±0.01	0.353±0.01
EventADL	**0.031±0.01**	**0.104±0.01**	**0.017±0.01**	**0.096±0.03**	**0.022±0.01**
ESP-only	0.001±0.00	0.004±0.00	0.007±0.00	0.006±0.02	0.006±0.02
EFP-only	0.030±0.01	0.100±0.01	0.010±0.01	0.090±0.02	0.016±0.01

(b) RCL Runtime (in seconds).

Method	Falcon	Flask	Live	OUT	AVA
BARO [42]	0.120±0.01	0.135±0.01	0.290±0.03	0.009±0.00	0.010±0.00
ϵ-Diagnosis [50]	2.320±3.25	2.470±3.15	1.620±2.02	1.130±0.12	0.160±0.05
Groot [59]	9.797±0.11	13.81±0.22	8.236±0.03	0.357±0.01	0.760±0.09
RCD [21]	15.78±36.1	10.79±24.1	9.530±17.9	0.190±0.02	81.21±0.67
CausalAI [5]	81.06±52.7	63.35±34.9	73.73±44.5	2.580±0.01	1.460±0.98
CausalRCA [63]	185.7±296	189.8±299	232.1±341	55.95±4.67	54.66±4.47
DeepHunt [53]	30.10±0.31	33.88±0.48	14.24±0.31	4.100±0.14	243.0±1.90
TVDiag [62]	66.90±0.22	73.10±0.00	85.11±0.00	-	-
KPIRoot [18]	554.9±0.56	907.4±27.2	445.3±13.5	11.37±0.29	600.21±0.84
EventADL	**2.366±0.06**	**4.319±0.04**	**2.420±0.04**	**0.130±0.00**	**0.034±0.00**

5.7.2 Root Cause Localization. As presented in Table 4 (b), **EventADL can localize the root cause of anomalies within seconds.** It takes 2.4, 4.3, and 2.4 seconds to localize the root causes in the Falcon, Flask, and Live datasets, respectively. We observe that the runtime is split fairly evenly between the construction of the intervention graph and the random walk (N=100). We observe that BARO is the fastest RCL method as it is a simple statistical approach that only considers three values (median, IQR, and maximum value) across all time series, making it very efficient. On the other hand, causal inference-based methods (CausalAI, CausalRCA, RCD) are time-consuming. These methods construct causal graphs between time series, which involves computationally expensive operations such as calculating correlations between all possible pairs of time series.

5.7.3 Scalability of EventADL. In this section, we evaluate the scalability of three components in EventADL when handling large event streams. We deploy EventADL on a machine with 12 vCPUs and 36GB RAM, and measure its runtime when processing event streams from deployed systems with different scales. Figure 6 presents the runtimes of EventADL across different scales.

We observe that EventADL can support real-time monitoring. Specifically, ESPs and EFPs can detect anomalies at rates of 100K events/s. Recall from our real-world analysis (Figure 2) that the number of

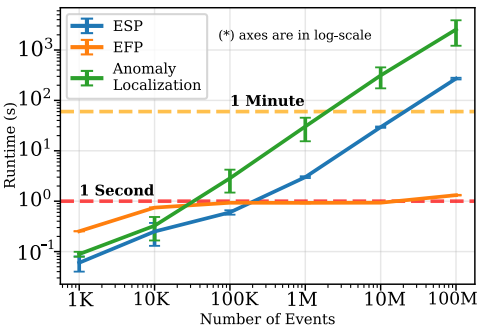


Fig. 6. Scalability of EventADL.

events per incident has a median of 1K and a 3^{rd} quartile of 100K. After detection, EventADL takes less than 1 minute to localize the root cause with up to 1M events. In addition, ESP and RCL exhibit linear growth with the number of events. This is expected, as ESP must scan the entire stream for pointwise anomaly detection and the construction of the Intervention Graph for RCL also require similar scanning. Nevertheless, both remain efficient at scale. Notably, EFP runtime remains nearly constant. Its runtime ranges from 0.25s at 1K events to only 1.3s at 100M events. The EFP module scales very favorably because EFP operates on event-based time series rather than raw events, and we observe that the number of unique time series extracted does not grow proportionally with the number of events.

5.8 RQ4: Ablation Study

We conduct ablation experiments to better understand the contribution of each component in EventADL. First, we examine how each component contributes to the overall anomaly detection performance, then we present empirical evidence supporting our choices in EFP design (magnitude-based over shape-based) and ESP choices (HyGLAD vs Drain). Note that the RCL component is already minimal, with no parts to isolate or remove.

5.8.1 Anomaly Detection. The results in Table 2 show that *ESP-only* and *EFP-only* achieve high performance and positively contribute to the overall performance of EventADL. We can observe that EFPs outperform other statistical methods: like *EFP-only*, NSigma and CUSUM also detect anomalies in frequencies of ESPs, yet *EFP-only* has a higher F1-score on average across all five datasets. We also observe that *ESP-only* and *EFP-only* have high recall on most datasets. This is because incidents often manifest anomalies across multiple dimensions, both pointwise and frequency-based anomalies. While our analysis in Section 3 shows that some incidents only manifest anomalies along a single dimension, there may exist undetected anomalies in other dimensions that remain unreported. To further assess how ESP and EFP complement each other, we randomly injected 20% anomalies into the historical (training) period for *ESP-only (C)* and *EFP-only (C)*. Naturally, both miss the anomalies injected in training, resulting in recall scores of only ≈80% each. However, their combination—*EventADL (C)*—achieves 96% recall, since *EFP-only (C)* is able to recover 80% of the anomalies missed by *ESP-only (C)*. This experiment shows that combining ESP and EFP helps detect subtle anomalies that may occur in only one dimension. We further show that our adaptation mechanism effectively reduces false positives while maintaining high recall. In dynamic cloud systems, without adaptation to system evolution, ESP and EFP may continuously flag false anomalies. Notably, *ESP-only* and *EFP-only*, when run without adaptation, achieve precision scores of 0.72 and 0.79 on the Falcon dataset, already outperforming most baselines. Our EventADL, which combines ESP, EFP, and adaptation, further reduces false alarms, raising precision to 0.82. Importantly, when encountering system evolution for the first time, ESP and EFP alone cannot distinguish legitimate evolution from anomalies, and will report both. With RCL, however, EventADL enables operators to identify the true sources of anomalies, discard false alarms, and trigger adaptation.

5.8.2 Shape-based vs Magnitude-based EFP. In this ablation, we compare our magnitude-based EFP with the shape-based variant [26, 32] to assess their impact on the OUT dataset. As shown in Figure 7, our magnitude-based EFP outperforms the shape-based approach with a substantially higher F1-score (0.741 vs. 0.155), demonstrating its effectiveness in detecting anomalies in event frequency (Section 4.3). Moreover, it achieves a 9× speedup in runtime (74.98ms vs. 664.08ms)

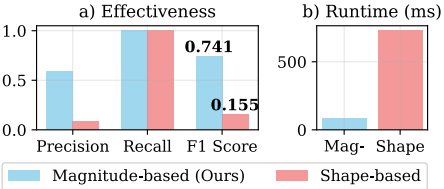


Fig. 7. Magnitude-based vs. shape-based EFP.

because our method does not require preprocessing the time series to match shapes. These results indicate that our magnitude-based subsequence comparison is better suited for event data.

5.8.3 HyGLAD-based vs Drain-based ESPs. As discussed in Section 4.2, EventADL can use different methods to learn ESPs. In this section, we replace HyGLAD [15] with Drain [20] to examine how the performance varies. Since Drain operates on unstructured logs, we implement two variants of event-to-log conversion. In the first variant, *EventADL(D1)*, we flatten each structured event into a log string. In the second variant, *EventADL(D2)*, we extract key fields (time, actor, operation, resource) from the events and construct log entries in the format: "`time=<time> actor=<actor> operation=<ops> resources=<res>`". Drain then learns the log templates, which we use as ESPs.

The experimental results presented in Table 2 show that both variants of EventADL using Drain for learning ESPs still outperform most baselines consistently across all datasets, demonstrating robustness when using different methods to learn ESPs. However, as discussed in Section 4.2, Drain may over-generalize some specific patterns on complex datasets (see Table B1 in the supplementary material), as it does not take into account the relationships between system entities, resulting in missed anomalies. In the Live dataset, EventADL's recall drops from 100% to 93% when using Drain to learn ESPs instead of HyGLAD, showing that HyGLAD is more suitable for event data.

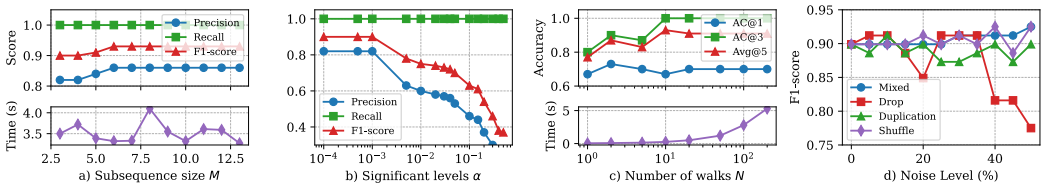


Fig. 8. Robustness analysis of EventADL w.r.t. different parameters and noise levels on the Falcon dataset.

5.9 RQ5: Robustness of EventADL

5.9.1 Robustness to Parameter Settings. In this section, we conduct a robustness analysis to understand how the performance of EventADL varies under different parameter settings. We refer readers to [15] for information about parameters for ESP. EFP has two parameters: (1) the subsequence size M, and (2) the significance level α. RCL has one parameter: the number of walks N used in the time-aware random walk (Algorithm 2).

We first vary the subsequence size M from 3 to 13 to see how EventADL performs with both smaller and larger subsequences. Second, we vary the significance level α from 10^{-4} to 0.05 to examine the sensitivity of EventADL. Third, we vary the number of walks N used in RCL from 1 to 200 to measure changes in performance. Due to space constraints, we only conduct these experiments on the Falcon dataset. The results are presented in Figure 8a, b, and c.

We observe that EventADL maintains stable performance across all values of M. However, shorter subsequences (e.g., $M < 6$) cause a slight drop in precision (from 0.86 to 0.82), as they may fail to capture more complex frequency patterns. Second, we find that EventADL becomes more sensitive as α increases (i.e., detecting more anomalies), resulting in lower precision and F1-scores. For example, precision drops from 0.82 to 0.23 as α increases from 10^{-4} to 0.05, reflecting a higher number of false positives. This finding suggests that a smaller α (e.g., 10^{-3}) is preferred to avoid over-triggering alarms. Third, we observe that very small values of N (e.g., 1 or 2) yield suboptimal localization performance due to limited exploration on the Intervention Graph. Accuracy improves significantly and stabilizes once $N \geq 10$, although runtime increases linearly. For instance, Avg@5 improves from 0.77 (at $N = 1$) to 0.91 (at $N = 20$), but runtime also increases from 0.05s to 0.5s. In practice, EventADL can execute RCL continuously, therefore, operators receive an initial ranked list of root causes that becomes progressively refined as additional nodes are traversed.

5.9.2 Robustness to Noise. We add noise to training event data by randomly (1) dropping, (2) duplicating, (3) shuffling, and (4) mixing these three, with intensity ranging from 1% and 50% of the total events. Figure 8d show that EventADL maintains stable performance under these noise conditions. However, it degrades under event dropping, as fewer events reduce the quality of learned patterns (e.g., F1 = 0.775 at 50% drop). These results show that our EventADL is resilient to imperfect training data–a critical property for deployment in noisy, evolving cloud environments.

5.10 Generalizability of EventADL to Non-Event Data

EventADL is developed specifically for event data, where ESP and RCL are designed to leverage structured information encoded in events such as actors, operations, and resources (Section 2.1). Therefore, ESP and RCL may not generalize to other data sources (e.g., metrics, logs, traces) as they do not contain this required information (see Figure 1). Nevertheless, EFP detects frequency-based anomalies and can be adapted to detect anomalies from other data sources. To assess the generalizability of EFP on non-event datasets, we evaluate its performance on three public benchmarks: Eadro [24], GAIA [11], and AIOps21 [35]. These datasets contain metrics, logs, and traces, which are fundamentally different from the structured event data. Eadro [24] provides two datasets with collected metrics, logs, and traces from two microservice systems, namely Train Ticket (Eadro TT) and Social Network (Eadro SN). Eadro TT has 27 services with 81 fault cases, and Eadro SN has 12 services with 36 fault cases. GAIA [11] provides the MicroSS-Companion dataset containing 216 labeled time series with injected anomalies across 7 anomaly types (e.g., changepoint, periodic). AIOps21 [35] is a dataset from the AIOps Challenge 2021 with 141 fault cases across 6 fault types.

To detect frequency-based anomalies using EFP, we follow existing works [19, 24, 54] to extract time series from the data (metrics, logs, traces). Similar to event-based anomaly detection (Section 4.3), EFP flags an anomaly if any of the time series exhibits frequency deviations during the test window. We apply EFP to all available time series in each test case. We benchmark EFP against baseline methods established in prior work on these datasets. For the Eadro datasets, we report results from the Eadro paper [24], including TraceAnomaly [31], MultimodalTrace [36], MS-RF-AD, MS-SVM-AD, MS-LSTM, and MS-DCC. For the GAIA dataset, we compare against baselines in the LogFormer paper [19], including SVM, DeepLog [13], LogAnomaly [33], PLELog [65], LogRobust [69], and LogFormer. For the AIOps21 dataset, we use baseline results from the ART paper [54], including ART, Eadro [24], and Hades [25]. Table 5 presents the results. We observe that:

Table 5. Performance comparison on public benchmarks: Eadro, GAIA, and AIOps21.

Method	Eadro TT			Eadro SN			Method	GAIA			Method	AIOps21		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score		Precision	Recall	F1-Score		Precision	Recall	F1-Score
TraceAnomaly [31]	0.486	0.414	0.589	0.539	0.468	0.636	SVM [19]	0.210	0.540	0.300	ART [54]	0.877	0.960	0.917
MultimodalTrace [36]	0.608	0.576	0.644	0.676	0.632	0.726	DeepLog [13]	0.180	0.820	0.310	Eadro [24]	0.767	0.935	0.842
MS-RF-AD [24]	0.817	0.705	0.971	0.773	0.866	0.700	LogAnomaly [33]	0.230	0.800	0.360	Hades [25]	0.867	0.868	0.868
MS-SVM-AD [24]	0.787	0.678	0.938	0.789	0.770	0.808	PLELog [65]	0.810	0.860	0.840	-	-	-	-
MS-LSTM [24]	0.967	0.997	0.940	0.948	0.959	0.937	LogRobust [69]	0.830	0.940	0.880	-	-	-	-
MS-DCC [24]	0.965	0.993	0.938	0.948	0.962	0.934	LogFormer [19]	0.890	0.980	0.930	-	-	-	-
EventADL	**0.745**	**0.975**	**0.845**	**0.857**	**1.000**	**0.923**	**EventADL**	**1.000**	**0.825**	**0.904**	**EventADL**	**0.870**	**0.930**	**0.899**

(1) EFP achieves high recall on Eadro datasets for detecting frequency-based anomalies. EFP achieves F1 of 84.5% and 92.3% on Eadro TT and Eadro SN, respectively, with recall scores of 97.5% and 100%. These datasets contain simple fault patterns as identified in [14], enabling high recall. MS-LSTM [24] and MS-DCC [24], deep learning baselines, achieve higher F1 scores of 94.0% and 93.8% on Eadro TT, and 93.7% and 93.4% on Eadro SN. However, these methods require training data with labeled failure samples, while EFP operates in a fully unsupervised manner.

(2) EFP maintains competitive performance on GAIA and AIOps21. On GAIA [11], EFP achieves F1 of 90.4% with 100% precision. EFP performs best on magnitude-based anomalies, achieving F1 of 98.5% on changepoint anomalies and 96.9% on partially stationary patterns. LogFormer [19],

a transformer-based method, achieves 93.0% F1. On AIOps21 [35], EFP achieves F1 of 89.9% with balanced precision (87.0%) and recall (93.0%), outperforming Eadro (84.2%) and Hades (86.8%), while ART [54] achieves 91.7% F1. We note that some GAIA anomalies are not visible upon manual inspection [8], and a recent evaluation [14] excluded GAIA due to ground-truth inconsistencies.

6 Threats to Validity

We assess potential threats to the validity of our work, following the guidelines outlined by Wohlin et al. [61]. The **construct validity** primarily concerns the hyperparameter settings and evaluation metrics. To mitigate this, we use established evaluation metrics and adopt recommended configurations from previous works [9, 42, 44, 45]. Another threat lies in the use of ESPs, which may be susceptible to adversarial evasion (e.g., attackers may mimic normal patterns to bypass detection). However, such evasive behavior likely triggers other consequences, which will eventually be detected by our framework. The **internal validity** stems from potential implementation bugs that could affect result reliability. We mitigate this by using well-maintained Python libraries, extensive testing, and repeating each experiment multiple times to ensure consistency. The **conclusion validity** stems from our benchmark datasets not covering the full range of anomaly types. While we base our incident reproduction on a systematic analysis of 520 real-world incident reports, certain scenarios require manual intervention because they fall outside EventADL's design scope. For anomaly detection, issues that do not manifest through event data cannot be detected. For example, a hardware fault (e.g., disk full) causing a node crash may not generate events. Similarly, performance degradation captured only in metrics (e.g., increased latency) without corresponding event signatures would be missed. For RCL, interventions with low connectivity in the Intervention Graph may receive fewer random walk visits than unrelated high-activity nodes, as discussed in Section 5. Nevertheless, EventADL still narrows the search space by identifying affected resources and recent interventions, enabling operators to extend their investigation. The **external validity** concerns the generalizability of our findings. In this study, we deployed our method in real cloud systems and evaluated against real incident data, grounding evaluation in realistic settings.

7 Conclusion

We present EventADL, the first open-box anomaly detection and RCL framework designed for event data in cloud systems. Our real-world incident analysis provides the empirical foundation for this work, revealing that event-based anomalies manifest through Event Type, Event Value, and Event Frequency, and their root causes require tracing intervention chains. Guided by these findings, EventADL detects pointwise anomalies through *Event Semantic Patterns (ESPs)* and frequency-based anomalies through *Event Frequency Patterns (EFPs)*, and localizes root causes by constructing an *Intervention Graph* and performing a time-aware random walk. Our evaluation on three benchmark systems and two real-world incidents demonstrates that EventADL achieves F1-scores of at least 90% for anomaly detection and 100% top-3 accuracy for RCL. We further show that EFP generalizes to non-event data, achieving competitive performance on three public benchmarks. We release our event datasets to facilitate future research on event-based ADL.

Data Availability

The implementation of EventADL, the three experimental datasets, and the supplementary materials are available on Zenodo at https://zenodo.org/records/19433493 [40].

Acknowledgments

This work was done during an internship at Amazon. Luan Pham was a PhD student at RMIT University under the Australian Research Council Discovery Project (DP220103044).

References

[1] Shan Ali, Chaima Boufaied, Domenico Bianculli, Paula Branco, and Lionel Briand. 2025. A Comprehensive Study of Machine Learning Techniques for Log-Based Anomaly Detection. *arXiv/2307.16714* (2025).

[2] Alibaba Cloud. 2025. *Alibaba Cloud ActionTrail.* https://www.alibabacloud.com/help/en/actiontrail/ ActionTrail monitors and records your Alibaba Cloud account activities; latest documentation updated February 12, 2025.

[3] Amazon Web Services. 2024. Understanding CloudTrail Events. https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-events.html. Accessed: 2025-06-06.

[4] Mohammad Ruhul Amin, Pranav Garg, and Baris Coskun. 2019. Cadence: Conditional anomaly detection for events using noise-contrastive estimation. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security.*

[5] Devansh Arpit, Matthew Fernandez, Itai Feigenbaum, Weiran Yao, Chenghao Liu, Wenzhuo Yang, Paul Josel, Shelby Heinecke, Eric Hu, Huan Wang, Stephen Hoi, Caiming Xiong, Kun Zhang, and Juan Carlos Niebles. 2023. Salesforce CausalAI Library: A Fast and Scalable Framework for Causal Analysis of Time Series and Tabular Data. (2023).

[6] Sergul Aydore, Baris Coskun, and Luca Melis. 2022. Detecting anomalous events from categorical data using autoencoders. US Patent 11,537,902.

[7] Ting Chen, Lu-An Tang, Yizhou Sun, Zhengzhang Chen, and Kai Zhang. 2016. Entity Embedding-Based Anomaly Detection for Heterogeneous Categorical Events. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence.* 1396–1403.

[8] Yuncong Chen. 2023. On injected anomalies. https://github.com/CloudWise-OpenSource/GAIA-DataSet/issues/11.

[9] Zhuangbin Chen, Jinyang Liu, Yuxin Su, Hongyu Zhang, Xiao Ling, Yongqiang Yang, and Michael R Lyu. 2022. Adaptive Performance Anomaly Detection For Online Service Systems Via Pattern Sketching. In *Proceedings of the 44th International Conference on Software Engineering.* 61–72.

[10] Qian Cheng, Doyen Sahoo, Amrita Saha, Wenzhuo Yang, Chenghao Liu, Gerald Woo, Manpreet Singh, Silvio Saverese, and Steven CH Hoi. 2023. AI for IT Operations (AIOps) on Cloud Platforms: Reviews, Opportunities and Challenges. *arXiv preprint arXiv:2304.04661* (2023).

[11] CloudWise-OpenSource. 2025. GAIA: Generic AIOps Atlas. https://github.com/CloudWise-OpenSource/GAIA-DataSet. CloudWise GAIA Dataset for AIOps.

[12] Baris Coskun, Wei Ding, and Luca Melis. 2022. Detecting anomalous events using autoencoders. US Patent 11,374,952.

[13] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of ACM SIGSAC conference on computer and communications security.*

[14] Aoyang Fang, Songhan Zhang, Yifan Yang, Haotong Wu, Junjielong Xu, Xuyang Wang, Rui Wang, Manyi Wang, Qisheng Lu, and Pinjia He. 2025. Rethinking the Evaluation of Microservice RCA with a Fault Propagation-Aware Benchmark. arXiv:2510.04711 [cs.SE] https://arxiv.org/abs/2510.04711

[15] Margarida Ferreira, Victor Nicolet, Luan Pham, Joey Dodds, Daniel Kroening, Ines Lynce, and Ruben Martins. 2025. Hypergraph-Guided Regex Filter Synthesis for Event-Based Anomaly Detection. arXiv:2509.06911 [cs.SE]

[16] Google Cloud. 2024. Cloud Audit Logs Overview. https://cloud.google.com/logging/docs/audit. Accessed: 2025-06-06.

[17] Wenwei Gu, Jiazhen Gu, Jinyang Liu, Zhuangbin Chen, Jianping Zhang, Jinxi Kuang, Cong Feng, Yongqiang Yang, and Michael R Lyu. 2025. ADAMAS: Adaptive Domain-Aware Performance Anomaly Detection in Cloud Service Systems. In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering.* 911–923.

[18] Wenwei Gu, Xinying Sun, Jinyang Liu, Yintong Huo, Zhuangbin Chen, Jianping Zhang, Jiazhen Gu, Yongqiang Yang, and Michael R Lyu. 2024. Kpiroot: Efficient monitoring metric-based root cause localization in large-scale cloud systems. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE).* IEEE, 403–414.

[19] Hongcheng Guo, Jian Yang, Jiaheng Liu, Jiaqi Bai, Boyang Wang, Zhoujun Li, Tieqiao Zheng, Bo Zhang, Junran Peng, and Qi Tian. 2024. Logformer: A pre-train and tuning pipeline for log anomaly detection. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 38. 135–143.

[20] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS).* IEEE, 33–40.

[21] Azam Ikram, Sarthak Chakraborty, Subrata Mitra, Shiv Saini, Saurabh Bagchi, and Murat Kocaoglu. 2022. Root cause analysis of failures in microservices through causal discovery. *Advances in Neural Information Processing Systems* 35 (2022), 31158–31170.

[22] Max Landauer, Florian Skopik, and Markus Wurzenberger. 2024. A critical review of common log data sets used for evaluation of sequence-based anomaly detection techniques. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 1354–1375.

[23] Van-Hoang Le and Hongyu Zhang. 2021. Log-based anomaly detection without log parsing. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 492–504.

[24] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R Lyu. 2023. Eadro: An end-to-end troubleshooting framework for microservices on multi-source data. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE).* IEEE, 1750–1762.

[25] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, Yongqiang Yang, and Michael R Lyu. 2023. Heterogeneous anomaly detection for software systems via semi-supervised cross-modal attention. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1724–1736.

[26] Daesoo Lee, Sara Malacarne, and Erlend Aune. 2024. Explainable time series anomaly detection using masked latent generative modeling. *Pattern Recognition* 156 (2024), 110826.

[27] Liqun Li, Xu Zhang, Shilin He, Yu Kang, Hongyu Zhang, Minghua Ma, Yingnong Dang, Zhangwei Xu, Saravan Rajmohan, Qingwei Lin, et al. 2023. Conan: Diagnosing batch failures for cloud systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 138–149.

[28] Mingjie Li, Zeyan Li, Kanglin Yin, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2022. Causal Inference-Based Root Cause Analysis for Online Service Systems with Intervention Recognition. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '22, Vol. 1)*. Association for Computing Machinery, New York, NY, USA, 3230–3240. doi:10.1145/3534678.3539041

[29] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. 2020. Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 92–103.

[30] Boyang Liu, Ding Wang, Kaixiang Lin, Pang-Ning Tan, and Jiayu Zhou. 2021. RCA: A Deep Collaborative Autoencoder Approach for Anomaly Detection. In *IJCAI*. ijcai.org, 1505–1511.

[31] Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, Wenman Xue, et al. 2020. Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 48–58.

[32] Yue Lu, Renjie Wu, Abdullah Mueen, Maria A Zuluaga, and Eamonn Keogh. 2022. Matrix profile XXIV: scaling time series anomaly detection to trillions of datapoints and ultra-fast arriving data streams. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1173–1182.

[33] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. 2019. LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *IJCAI*, Vol. 19. 4739–4745.

[34] Microsoft Corporation. 2024. Ingest Events from Azure Event Hubs into Azure Monitor Logs. https://learn.microsoft.com/en-us/azure/azure-monitor/logs/ingest-logs-event-hub. Accessed: 2025-06-06.

[35] AIOps Nankai. 2021. AIOps 2021 Challenge Dataset. https://www.aiops.cn/gitlab/aiops-nankai/data/trace/aiops2021/.

[36] Sasho Nedelkoski, Jorge Cardoso, and Odej Kao. 2019. Anomaly detection from system tracing data using multimodal deep learning. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 179–186.

[37] Hiep Nguyen, Yongmin Tan, and Xiaohui Gu. 2011. PAL: Propagation-aware Anomaly Localization for Cloud Hosted Distributed Applications. In *Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*. 1–8.

[38] Open Cybersecurity Schema Framework. 2022. Open Cybersecurity Schema Framework (OCSF). https://github.com/ocsf Accessed: 2025-08-04.

[39] OpenTelemetry. 2025. Semantic Conventions for Events. https://opentelemetry.io/docs/specs/semconv/general/events/. Accessed: 2025-06-06.

[40] Luan Pham. 2026. *Artifacts of "EventADL: Open-Box Anomaly Detection and Localization Framework for Events in Cloud-Based Service Systems"*. doi:10.5281/zenodo.19433493

[41] Luan Pham. 2026. Graph-Free Root Cause Analysis. *arXiv preprint arXiv:2601.21359* (2026).

[42] Luan Pham, Huong Ha, and Hongyu Zhang. 2024. BARO: Robust root cause analysis for microservices via multivariate bayesian online change point detection. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 2214–2237.

[43] Luan Pham, Huong Ha, and Hongyu Zhang. 2024. Root Cause Analysis for Microservice System based on Causal Inference: How Far Are We?. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 706–715.

[44] Luan Pham, Huong Ha, Xiuzhen Zhang, and Hongyu Zhang. 2026. TORAI: Multi-Source Root Cause Analysis for Blind Spots in Microservice Service Call Graph. *Proceedings of the ACM on Software Engineering* 3, FSE (2026).

[45] Luan Pham, Hongyu Zhang, Huong Ha, Flora Salim, and Xiuzhen Zhang. 2025. RCAEval: A Benchmark for Root Cause Analysis of Microservice Systems with Telemetry Data. In *Companion Proceedings of the ACM on Web Conference 2025*.

[46] Chen Qiu, Timo Pfrommer, Marius Kloft, Stephan Mandt, and Maja Rudolph. 2021. Neural Transformation Learning for Deep Anomaly Detection Beyond Images. In *Proceedings of Machine Learning Research*, Vol. 139. 8703–8714.

[47] Rui Ren, Jingbang Yang, Linxiao Yang, Xinyue Gu, and Liang Sun. 2024. SLIM: a Scalable Light-weight Root Cause Analysis for Imbalanced Data in Microservice. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*. 328–330.

[48] Devjeet Roy, Xuchao Zhang, Rashi Bhave, Chetan Bansal, Pedro Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. 2024. Exploring LLM-based Agents for Root Cause Analysis. *arXiv preprint arXiv:2403.04123* (2024).

[49] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. 2018. Deep one-class classification. In *International conference on machine learning*. PMLR.
[50] Huasong Shan, Yunpeng Zhang, Yuan Chen, Xiao Xiao, Haifeng Liu, Xiaofeng He, Min Li, and Wei Ding. 2019. ϵ-Diagnosis: Unsupervised and real-time diagnosis of small-window long-tail latency in large-scale microservice platforms. *Proceedings of the World Wide Web Conference, WWW 2019* (2019), 3215–3222.
[51] Tom Shenkar and Lior Wolf. 2022. Anomaly detection for tabular data with internal contrastive learning. In *International Conference on Learning Representations*.
[52] Jacopo Soldani and Antonio Brogi. 2022. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. *ACM Computing Surveys (CSUR)* 55, 3 (2022), 1–39.
[53] Yongqian Sun, Zihan Lin, Binpeng Shi, Shenglin Zhang, Shiyu Ma, Pengxiang Jin, Zhenyu Zhong, Lemeng Pan, Yicheng Guo, and Dan Pei. 2025. Interpretable failure localization for microservice systems based on graph autoencoder. *ACM Transactions on Software Engineering and Methodology* 34, 2 (2025), 1–28.
[54] Yongqian Sun, Binpeng Shi, Mingyu Mao, Minghua Ma, Sibo Xia, Shenglin Zhang, and Dan Pei. 2024. Art: A unified unsupervised framework for incident management in microservice systems. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 1183–1194.
[55] Wil Van der Aalst, Ton Weijters, and Laura Maruster. 2004. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering* 16, 9 (2004), 1128–1142.
[56] Jeremy Wadhams. n.d.. JSONLogic: A lightweight, safe way to share logic between systems. https://jsonlogic.com/.
[57] Dongjie Wang, Zhengzhang Chen, Yanjie Fu, Yanchi Liu, and Haifeng Chen. 2023. Incremental causal graph learning for online root cause analysis. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*. 2269–2278.
[58] Hu Wang, Guansong Pang, Chunhua Shen, and Congbo Ma. 2020. Unsupervised Representation Learning by Predicting Random Distances. In *IJCAI*. ijcai.org, 2950–2956.
[59] Hanzhang Wang, Zhengkai Wu, Huai Jiang, Yichao Huang, Jiamu Wang, Selcuk Kopru, and Tao Xie. 2021. Groot: An Event-graph-based Approach for Root Cause Analysis in Industrial Settings. *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering* (2021), 419–429.
[60] Yidan Wang, Zhouruixing Zhu, Qiuai Fu, Yuchi Ma, and Pinjia He. 2024. MRCA: Metric-level root cause analysis for microservices via multi-modal data. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 1057–1068.
[61] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, Anders Wesslén, et al. 2024. *Experimentation in Software Engineering*. Springer.
[62] Shuaiyu Xie, Jian Wang, Hanbin He, Zhihao Wang, Yuqi Zhao, Neng Zhang, and Bing Li. 2026. TVDiag: A Task-oriented and View-invariant Failure Diagnosis Framework for Microservice-based Systems with Multimodal Data. *ACM Transactions on Software Engineering and Methodology* 35, 2 (2026), 1–39.
[63] Ruyue Xin, Peng Chen, and Zhiming Zhao. 2023. CausalRCA: Causal inference based precise fine-grained root cause localization for microservice applications. *Journal of Systems and Software* 203 (2023), 111724.
[64] Hongzuo Xu, Guansong Pang, Yijie Wang, and Yongjun Wang. 2023. Deep Isolation Forest for Anomaly Detection. *IEEE Trans. Knowl. Data Eng.* 35, 12 (2023), 12591–12604.
[65] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. 2021. Semi-supervised log-based anomaly detection via probabilistic label estimation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1448–1460.
[66] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. 2023. Nezha: Interpretable fine-grained root causes analysis for microservices on multi-modal observability data. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 553–565.
[67] Jun Zengy, Xiang Wang, Jiahao Liu, Yinfang Chen, Zhenkai Liang, Tat-Seng Chua, and Zheng Leong Chua. 2022. Shadewatcher: Recommendation-guided cyber threat analysis using system audit records. In *2022 IEEE symposium on security and privacy (SP)*. IEEE, 489–506.
[68] Chenxi Zhang, Zhen Dong, Xin Peng, Bicheng Zhang, and Miao Chen. 2024. Trace-based multi-dimensional root cause localization of performance issues in microservice systems. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–12.
[69] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 807–817.
[70] Lecheng Zheng, Zhengzhang Chen, Jingrui He, and Haifeng Chen. 2024. MULAN: multi-modal causal structure learning and root cause analysis for microservice systems. In *Proceedings of the ACM Web Conference 2024*. 4107–4116.

Received 2025-09-11; accepted 2026-03-24